Verifiable Multi-Party Computation with Perfectly Private Audit Trail

Édouard Cuvelier and Olivier Pereira

Univeristé catholique de Louvain
ICTEAM/ELEN – Crypto Group
1348 Louvain-la-Neuve – Belgium

Abstract. We propose an efficient protocol for the evaluation of functions getting their inputs from multiple parties in a way that guarantees the result correctness. In our setting, a worker is trusted with the confidentiality of the inputs and, given this assumption, our protocol guarantees perfect privacy to the clients.
Our protocol offers an interesting middle ground between traditional verifiable computation protocols, that usually do not come with privacy guarantees and focus on one or a small number of clients, and secure multi-party computation protocol that distribute the privacy trust between a number of parties, at the cost of much more expensive protocols (especially for NP functions and functions that do not admit an efficient static circuit representation) and a demanding infrastructure of independently managed servers interacting in multiple rounds. By contrast, our protocol is single-pass: the clients submit their inputs asynchronously, and everyone can collect the result at any later time.
We present three unrelated applications of our technique: solving a system of linear equations, an auction scheme and the search of the shortest path in a shared graph. These examples illustrate the ease of use and the advantage in terms of complexity of our approach. We made a prototype implementation that illustrates the practicality of our solution.

1 Introduction

We investigate the well-known problem of a set of clients holding private inputs and looking for an efficient solution for the evaluation of a function of these inputs.

In most practical cases, e.g., auctions, health information management systems, benchmarking services, or cloud services in general, this problem is handled by delegating all the confidential inputs to a trusted third party, or worker, who is in charge of computing and distributing the output of the computation to the clients. The trust encompasses two different aspects: the correctness of the computation, and the confidentiality of the inputs.

Several important lines of work addressed these two forms of trust in different settings. Secure Multi-Party Computation (SMC) addresses the confidentiality issue by distributing the computation between several workers (who can also be the

clients) and often also addresses the correctness aspect through zero-knowledge proofs. Despite tremendous improvements in terms of efficiency that happened during the last few years, e.g., through the "SPDZ" protocol of Damgård et al. [1] and its publicly verifiable improvement proposed by Baum et al. [2], there remain important practical obstacles to the broad deployment of these techniques. First, a large computational and communication overhead seems inevitable due to the need to perform the whole computation process step by step in a distributed fashion. This concern becomes even stronger when the function that needs to be evaluated does not admit a simple static circuit representation, as it is the case when efficient solutions require non-uniform data-dependent branching (see examples from Aly et al. for instance [3]). Besides, from an organizational point of view, it is often difficult to obtain that the various workers independently deploy and manage servers on a common high-speed network. For example, the sugar beet auction in Denmark [4] was performed between three parties representing farmers, buyers and the SMC project promoters, who were trusted by all the clients. This is also the case in some cryptographic voting systems such as Helios [5] where, despite the simplicity of the function that is evaluated (a sum), the tallying is performed by a small set of trustees sharing the private key of a distributed encryption scheme

As a second line of work, Verifiable Computation (VC) addresses this infrastructure problem by investigating solutions based on a single worker, who could be a cloud service provider. For instance, the "Pinocchio" protocol proposed by Parno et al. [6], and its refinement "Gepetto" [7] are highly efficient solutions that offer public verifiability in a single client-worker setting. However the protocol does not aim at providing privacy of the inputs. Even more efficient than "Pinocchio", Backes et al. [8] developed a three-party protocol where a worker is requested to prove computations to a client over authenticated data received from a single trusted source. The construction of Parno has also been used by Zhang et al. [9] in "Alitheia", a single-client verifiable computation system for graph problems such as the shortest path studied in this paper.

The addition of confidentiality constraints in VC, that is, considering a single worker that is not trusted for confidentiality, has been formalized by Gennaro et al. [10] in the single client setting, then extended by Choi et al. [11] who achieve non-interactive multi-client verifiable computation by relying on garbled circuits, oblivious transfer and fully homomorphic encryption. In the follow-up works of Goldwaser et al. [12] and Gordon et al. [13], the solution uses functional encryption and indistinguishability obfuscation. These works have a largely theoretical flavor, and do not provide any concrete efficiency analysis.

Our Contributions. Our setting aims at practical solutions and focuses on an interesting middle-ground between the two forms of VC described above. Our unique worker performs verifiable computation, and the proof of correctness of its outputs preserves the privacy of the inputs (in an information theoretic way). Still, our worker is trusted to preserve the privacy of the inputs.

Trusting the worker for confidentiality has serious practical benefits: the worker is able to compute efficiently on cleartext data, and the function evaluation therefore does not come with any overhead. The proof of correctness is based on cryptographic primitives but, in many practical applications, is considerably cheaper than the computation itself: these applications include most problems in NP, and even simple standard problems like sorting. Besides, it makes it possible to perform the computation using highly sophisticated algorithms, circumventing issues related to functions that do not have a static circuit representation, and allowing the worker to use its own proprietary and confidential solution.

The level of interaction in our protocol is minimal: the clients submit their inputs to the worker as a single message, and the result and a single publicly verifiable proof is made available at the end of the computation. This makes our solution practical even for applications based on a web interface, which clients could use to submit their input, and later retrieve the outcome of the computation.

We define the security properties of our scheme through ideal functionalities for secure function evaluation. Our protocol guarantees the correctness of the output, even if the worker is corrupted. Furthermore, our protocol guarantees information theoretic privacy if the worker is honest.

We illustrate our technique via three test applications: solving a system of linear equations, electronic auctions and finding the shortest path in a graph. Finally, we give some insight on the performances obtained for these applications by a prototype implementation realized in Python.

Related works. Besides the works described above, and very close to our technique, Rabin et al. [14, 15] present a secrecy-preserving proof of correctness scheme for the evaluation of any function with straight line computation through an agreed public circuit. Indeed, similar to what is done in this paper, they propose to perform the proof of correctness on the commitments on the inputs of the function. A parallel circuit is evaluated by a worker on the commitments and every operation is validated by a zero-knowledge proof of knowledge. While the schemes proposed there rely on symmetric cryptography and a split-value representation to perform cut-and-choose proofs, we show in this work better timing results as well as more compact proofs using homomorphic cryptography based on elliptic curves.

Outline. The structure of the paper is as follows: in Section 2 we describe the functionality and the security proofs. In Section 3, we present the building blocks needed for a generic implementation. Section 4 details the three test applications while Section 5 provides the technical information, the complexity study, and the timing measurements obtained from our prototype implementation.

2 Verifiable Multi-Party Function Evaluation

The ideal protocol. In this section, we specify our protocol in terms of an ideal functionality, following the notations and definitions of Canetti [16]. We will then require our protocol to offer the same security features as that functionality.

In this regard, let us consider a set of clients $\mathcal{C} = \{C_1, \cdots, C_n\}$. Each C_i has a private input $x_i \in I$, the input space. We define our *ideal functionality* $\mathcal{F}^f$ as a process that privately receives inputs from the clients and then evaluates the function $f : I^n \to O$ on these inputs (O is the output space), and outputs that result to all parties. So, correctness is always guaranteed by this functionality.

We consider two corruption models for our protocol, which lead to two flavors of our functionality. In the case of a honest-but-curious (also often called "passive") adversary $\mathcal{A}_p$, that is, an adversary who learns the internal state of the corrupted parties but lets them follow the protocol, the functionality $\mathcal{F}^f_{\mathcal{A}_p}$ also guarantees that the clients do not learn anything about each other's inputs (apart from what might be derived from the output of the function). In the case of an active adversary $\mathcal{A}_a$, correctness remains guaranteed, but the client's inputs are leaked to the adversary, and confidentiality is therefore not guaranteed anymore.

The ideal functionalities $\mathcal{F}^f_{\mathcal{A}_p}$ and $\mathcal{F}^f_{\mathcal{A}_a}$

---

1. Upon receiving $(\mathsf{Send}, C_i, x_i)$ from a client C_i or adversary $\mathcal{S}$, if $x_i \in I$, store x_i, otherwise abort. Then, in the case of
   - a *honest-but-curious adversary*, send C_i to adversary $\mathcal{S}$ and halt.
   - an *active adversary*, send (C_i, x_i) to adversary $\mathcal{S}$ and halt.
2. Upon receiving $\mathsf{Compute}$ from $\mathcal{S}$, evaluate $y := f(x_1, \cdots, x_n)$. Send y to every client C_i and $\mathcal{S}$, then halt.

The real protocol. We now turn to the design of our real-world protocol, that realizes the ideal functionalities.

We require our protocol to produce a perfectly private audit trail (PPAT) of its computation, that is, the privacy guarantees offered by our protocol will be perfect in the sense of information theory. For simplicity, we focus on the case of static corruption: corruption of parties happen before the beginning of the protocol, and not dynamically as the protocol executes.

We build the protocol Π^f_{PPAT} which realizes these functionalities in the real world in the presence of honest-but-curious and active adversaries. In this protocol, most of the work of the functionality is performed by an entity called the Worker $\mathcal{W}$. However, contrary to the ideal functionality, the worker can be corrupted by the adversary and, in the case of an active adversary, might be willing to evaluate a function different of f.

Designing a protocol that implements our functionality in the presence of an honest-but-curious adversary is fairly immediate: clients hand their inputs

to the worker through a secure channel (which can be realized by means of a CPA-secure encryption scheme), then the worker simply outputs the result.

The case of an active adversary is more demanding: in order to make sure that an invalid output of the worker will not be accepted by the clients, our protocol requires the worker to provide a proof of the correctness of the output. And, in order to ensure that every client receives the same result at the end of the protocol, this proof will be posted on a Public Bulletin Board **PB**, which maintains publicly available every input sent to him by any parties.

The most immediate way of building the proof would be to use zero-knowledge proofs computed from the encrypted inputs of the clients. This has two downsides, however. In terms of efficiency, secure encryption is length increasing, which will typically lead to more expensive proofs, since they need to apply to larger statements. In terms of security, the ciphertexts that will be needed to verify the proofs are only computationally hiding, meaning that the inputs of the client will eventually become available through data that are published as part of the protocol.

An alternative approach, which we adopt, is to work on perfectly hiding commitments: they do not need to be length increasing, potentially leading to more efficient proofs, and they do not cause any weakening of privacy. Perfectly hiding commitments can only be computationally binding, but this is a much lower concern, since the workers only have a relatively short period of time for producing their proofs, making any future computational breakthrough innocuous.

So, our protocol will rely on a commitment scheme, that is, a triple of algorithms $\Pi_{\mathsf{C}} := (\mathsf{Gen}_{\mathsf{C}}, \mathsf{Com}, \mathsf{Verify})$. Algorithm Com, on input x and public key cpk generated by $\mathsf{Gen}_{\mathsf{C}}$, produces d, o where d is the commitment and o is the opening value used afterwards to open the commitment through the Verify algorithm. The perfectly hiding property of the commitment scheme means that for any commitment d, we can find, for any value x in the input space, an opening o such that $\mathsf{Verify}(d, o, x)$ accepts. In other words, it implies that no single piece of information from x could be extracted from d. We also require the commitments to be computationally binding: it must be hard to produce a commitment d, two messages $x \neq x'$ and two opening values o, o' on which d can be opened.

As a first step in our protocol, the clients send a commitment $d_i \leftarrow \mathsf{Com}(x_i)$ on their private inputs x_i to **PB**. Along with the commitment d_i, each client must produce a proof denoted $\pi_{\mathsf{ver}}(d_i)$ that ensures non-malleability [17], under the form of a Σ-proof of knowledge (see [18]). Non-malleability will be used to enforce the independence of the inputs posted by the clients [18], preventing one client from choosing his input as a function of someone else's input, which could have devastating effects in some contexts (e.g., auctions). In some cases, the proof $\pi_{\mathsf{ver}}(d_i)$ can also be designed to be more than just a proof of knowledge: for instance, a Σ-proof of membership can be used to ensure the validity of x_i.

In parallel with the posting of these commitments and proofs, all clients submit an opening of their commitment to the worker $\mathcal{W}$, using a private channel. From these inputs, $\mathcal{W}$ posts the evaluation of f on **PB**. And, in order to capture an actively corrupted $\mathcal{W}$, we require that $\mathcal{W}$ also publishes a proof denoted

π_{cor} of the correctness of the evaluation y of f on the inputs. The key point is that the verification of π_{cor} relies on the commitments $d_1, \ldots, d_n$ posted by the clients on **PB**. In the general case, this verification involves the computation of a commitment d_y that must be a commitment on y computed from $d_1, \ldots, d_n$. With this requirement, an active adversary who would be willing to cheat during the function evaluation process would need to be able to break the binding property of the commitment scheme or the soundness of the proof π_{cor}.

Protocol Π^f_{PPAT} and formal security. The proofs we are referring to here are built from the notion of sigma (or Σ)-protocols [19]. In the following, we define relations R in a formal NP-language such as $R \subset \mathcal{L}_{\mathsf{NP}} \times W(s)$ where s is called the statement and $w \in W(s)$ a witness of s.

We rely on the Fiat-Shamir/Blum transformation [20, 21] to turn Σ-protocols into non-interactive zero-knowledge proofs of knowledges (NIZKPK). This heuristic relies on the existence of an efficient hash function which is used to create the challenge $\mathfrak{e}$ of the Σ-protocol. Careful attention must be paid on the choice of the values thrown into the hash function [18]. It is essential to hash at least the commitment generated in the Σ-protocol together with the statement s.

Definition 1 (Non-Interactive Zero-Knowledge Proof of Knowledge). *A* Non-Interactive Zero-Knowledge Proof of Knowledge π *for a relation R on an* NP*-language $\mathcal{L}_{\mathsf{NP}}$ is a couple of efficient algorithms* $(\mathsf{Prove}, \mathsf{Check})$ *such that:*

$\mathsf{Prove}(s, w)$*: on inputs $(s, w) \in R$, outputs a transcript* $\mathfrak{t}$.
$\mathsf{Check}(s, \mathfrak{t})$*: on inputs a statement s and a transcript* $\mathfrak{t}$*, outputs 0 or 1.*

where Prove *is probabilistic and* Check *is deterministic. The next properties hold:*

Completeness: $\forall (s, w) \in R, \forall \mathfrak{t} \leftarrow \mathsf{Prove}(s, w)$*, we have that* $\mathsf{Check}(s, \mathfrak{t}) = 1$ *with overwhelming probability.*

Soundness: *there exists a polynomial time extractor E such that, when E receives on inputs two valid transcripts $\mathfrak{t}, \mathfrak{t}'$ with respect to the same s, E returns a correct witness w.*

Perfect Zero-Knowledge: *there exists a probabilistic polynomial time simulator $\mathcal{M}$ that produces simulated transcripts that are perfectly indistinguishable from real transcripts produced through the* Prove *algorithm.*

The parties involved in the protocol Π^f_{PPAT} will publish on **PB** transcripts of NIZKPK to prove some relations. We describe in Section 3 the relations we aim to prove and how to build specific proofs for our applications.

The relations for the NIZKPK π_{cor} and $\pi_{\mathsf{ver}}(d_i)$ mentioned in Π^f_{PPAT} are defined as follows:

$$R^{\mathsf{cor}} := \{((y, d_1, ..., d_n), (x_1, o_1, ..., x_n, o_n)) | \quad y = f(x_1, ..., x_n) \bigwedge_i \mathsf{Verify}(d_i, o_i, x_i) = 1\}$$

$$R^{\mathsf{ver}} := \{(d, (x, o)) | \mathsf{Verify}(d, o, x) = 1 \wedge x \in I\}$$

where the algorithm $\mathsf{Verify}(d, o, x)$ of the commitment scheme returns 1 only if d is a commitment on x with opening o.

These proofs are published on **PB** and π_{cor} will be checked by each C_i at the end of the protocol to convince itself of the correctness of the function's computation. We are now ready to define protocol Π^f_{PPAT} which realizes functionalities $\mathcal{F}^f_{\mathcal{A}_p}$ and $\mathcal{F}^f_{\mathcal{A}_a}$ in the presence of $\mathcal{A}_p$ and $\mathcal{A}_a$ respectively.

Protocol Π^f_{PPAT}

Input: Each C_i has his private input $x_i \in I$ for $i = 1, \cdots, n$.
Output: Each C_i receives $y := f(x_1, \cdots, x_n)$.

1. Each C_i computes a perfectly hiding commitment on $x_i : d_i, o_i \leftarrow \mathsf{Com}(x_i)$ as well as a proof $\pi_{\mathsf{ver}}(d_i)$ on some property that x_i must meet. C_i publishes d_i and $\pi_{\mathsf{ver}}(d_i)$ on **PB**. Then, C_i sends x_i, o_i to $\mathcal{W}$ through the *secure channel*.
2. $\mathcal{W}$ runs $\mathsf{Check}_{\mathsf{ver}}$ of $\pi_{\mathsf{ver}}(d_i)$ on each d_i and aborts if one of the check is false. $\mathcal{W}$ runs the Verify algorithm on each triple (d_i, o_i, x_i) and aborts if one verification fails. Otherwise, on inputs $x_1, ..., x_n, o_1, ..., o_n$, $\mathcal{W}$ computes $y := f(x_1, ..., x_n)$ and a proof of correctness π_{cor} of the result. Then, $\mathcal{W}$ publishes y and π_{cor} on **PB**.
3. Each C_i runs $\mathsf{Check}_{\mathsf{ver}}$ of $\pi_{\mathsf{ver}}(d_j)$ on each d_j (for $j = 1, ..., n$ and $j \neq i$) and $\mathsf{Check}_{\mathsf{cor}}$ of π_{cor} on $(y, d_1, ..., d_n)$. If each verification accepts, then C_i accepts output y, otherwise C_i aborts.

We show in Section 3.3 how to build protocol Π^f_{PPAT} for any function f and how $\mathcal{W}$ can prepare the proof π_{cor}. Note that, in step 2, it is crucial for $\mathcal{W}$ to verify that the commitments on **PB** are consistent with the private inputs. Indeed, if a cheating client published a commitment that is not consistent with his private input, then $\mathcal{W}$ will not be able to provide a correct proof π_{cor} on the result since this proof is bound to the commitments. In Appendix A, we point out two modifications of the protocol that achieve slightly different functionalities. The first one keeps the outcome y secret so that the function evaluation can be used as a subroutine of a larger function without revealing intermediary results. The second one aims at encountering active adversaries that dynamically chooses the inputs of the corrupted clients as a function of the inputs of honest clients.

One can see that the security of the protocol in the presence of a passive adversary $\mathcal{A}_p$ rests on the fact that every piece of information present on **PB** is either perfectly hiding or zero-knowledge. However, in the presence of an active adversary $\mathcal{A}_a$ the privacy of the scheme is not guaranteed: a corrupted worker could disclose all the client inputs. Nevertheless, in this scenario, we assert that the verifiability property still stands. In other words, $\mathcal{A}_a$ could leak the private inputs but is not able to tamper with the correctness of the function evaluation.

Our first result shows that protocol Π^f_{PPAT}, executed with an ideal bulletin board, realizes the functionality $\mathcal{F}^f_{\mathcal{A}_p}$ in the presence of passive adversary $\mathcal{A}_p$, and has a perfectly private audit trail.

Theorem 1. *Let* $\Pi_{\mathsf{C}} := (\mathsf{Gen}_{\mathsf{C}}, \mathsf{Com}, \mathsf{Verify})$ *be a perfectly hiding commitment scheme and* π_{cor} *and* π_{ver} *be perfect zero-knowledge proofs. Then, for any set of corrupted clients, there is a simulator such that, for any environment, the views resulting from the following two situations are* ***identical****:*

- *interacting with the bulletin board, the clients and the worker playing the* Π^{f}_{PPAT} *protocol.*
- *interacting with the ideal functionality* $\mathcal{F}^{f}_{\mathcal{A}_p}$ *and the simulator.*

The view of the environment includes its accesses to the bulletin board (controlled by the simulator in the second case), submitting the input x_i *to the clients (or to* $\mathcal{F}^{f}_{\mathcal{A}_p}$*), and obtaining the outcome* y *in return.*

Proof. Informal. We proceed by a set of game hops to show that the view of the environment and the adversary is indistinguishable between the real execution of the protocol and the ideal execution simulating the functionality $\mathcal{F}^{f}_{\mathcal{A}_p}$. The key points are that

1. the commitments published on **PB** reveal no information whatsoever about the committed values.
2. the proofs of knowledge on **PB** are perfect zero-knowledge and cannot be used by an adversary to extract information.
3. the proof π_{cor} present on **PB** computationally ensures the result soundness.
4. the three first points combined form a perfectly private audit trail of the function evaluation that is computationally sound.

The complete proof can be found in Appendix B.1.

Our second result shows that Protocol Π^{f}_{PPAT}, executed with an ideal bulletin board, realizes the functionality $\mathcal{F}^{f}_{\mathcal{A}_a}$ in the presence of an active adversary $\mathcal{A}_a$ who controls the worker.

Theorem 2. *Let* $\Pi_{\mathsf{C}} := (\mathsf{Gen}_{\mathsf{C}}, \mathsf{Com}, \mathsf{Verify})$ *be a binding commitment scheme and* π_{cor} *and* π_{ver} *be computationally sound proofs. Then, for a corrupted worker and any set of corrupted clients, there is a simulator such that, for any environment, the views resulting from the two situations below are* ***indistinguishable****:*

- *interacting with the bulletin board, the clients and the corrupted worker playing the* Π^{f}_{PPAT} *protocol.*
- *interacting with the ideal functionality* $\mathcal{F}^{f}_{\mathcal{A}_a}$ *and the simulator.*

The view of the environment includes its accesses to the bulletin board (controlled by the simulator in the second case), submitting the input x_i *to the clients (or to* $\mathcal{F}^{f}_{\mathcal{A}_a}$*), and obtaining the outcome* y *in return, and every communication that the corrupted worker would make.*

Proof. Informal. The demonstration follows the same pattern of Theorem 1 with the major difference that the privacy of the inputs is no longer guaranteed due to the adversary ability to corrupt the worker. However, the soundness of the

proof π_{cor} remains ensuring the correctness of the result. We show that it is still essential that **PB** displays the computationally binding commitments of the clients. This condition enforces an adversary to produce a proof of knowledge on these commitments. As a result the audit trail present on **PB** is not perfectly hiding anymore but is still computationally sound.

The complete proof can be found in Appendix B.2.

3 Building Blocks for Perfectly Private Audit Trail

The interactions between the clients and the worker involve the exchange of private inputs and the publication on a Public Bulletin Board **PB** of some trail that will be used to perform further audit of the process. Depending on the properties one wants to achieve in different scenarios, we propose to use a primitive introduced by Cuvelier et al. in [22] called *commitment consistent encryption*. The primitive is a combination of an encryption scheme and a commitment scheme. It allows one to send its private inputs through a ciphertext while publicly committing on the same inputs for further public verification.

3.1 Commitment Consistent Encryption Scheme

A Commitment Consistent Encryption (CCE) scheme is a traditional public key encryption scheme that offers an extra feature: from any CCE ciphertext, it is possible to derive a commitment on the encrypted message, and the private key can also be used to obtain an opening of that commitment.

Definition 2 (CC Encryption). *A commitment consistent encryption scheme Π is a tuple of efficient algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{DerivCom}, \mathsf{Open}, \mathsf{Verify})$ *defined as follows:*

- $\mathsf{Gen}(1^\lambda)$*: Given a security parameter λ, output a triple* $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk})$*, respectively the public parameters, the public key and the secret key.*
- $\mathsf{Enc}_{\mathsf{pk}}(m)$*: Output a ciphertext c which is an encryption using the public key* pk *of a message m chosen in the plaintext space $\mathbf{M}$ defined by* pp*.*
- $\mathsf{Dec}_{\mathsf{sk}}(c)$*: From a ciphertext c, output a message m using the secret key* sk*.*
- $\mathsf{DerivCom}_{\mathsf{pk}}(c)$*: Output a commitment d from a ciphertext c using* pk*.*
- $\mathsf{Open}_{\mathsf{sk}}(c)$*: Output an auxiliary value o using the secret key* sk*. This auxiliary value can be considered as part of an opening for a commitment.*
- $\mathsf{Verify}_{\mathsf{pk}}(d, o, m)$*: From a message m, a commitment d with respect to key* pk *and an auxiliary value o, output a bit. This algorithm checks the validity of the opening (m, o) with respect to d and* pk*.*

It is implicit that pp *is given to each algorithm apart from* Gen*.*

Correctness*: for any* $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$*, any message $m \in \mathbf{M}$ and any ciphertext $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m)$, it holds with overwhelming probability in λ that* $\mathsf{Dec}_{\mathsf{sk}}(c) = m$ *and that* $\mathsf{Verify}_{\mathsf{pk}}(\mathsf{DerivCom}_{\mathsf{pk}}(c), \mathsf{Open}_{\mathsf{sk}}(c), \mathsf{Dec}_{\mathsf{sk}}(c)) = 1$*.*

The security properties that we can expect from the encryption part of the CCE scheme are the traditional ones. We refer to the work [22] for a more complete description of CCE scheme. This paper also presents a CCE scheme that is built from a Pedersen commitment and a Paillier encryption as well as two other optimal constructions of CCE based on elliptic curves.

In some settings such as e-voting, an homomorphic encryption scheme that allows threshold decryption is mandatory while in other settings, the encryption scheme could be superfluous when using a physically secure channel between the clients and the workers. In this last case, one may be just fine with a commitment scheme alone. However, in most cases, we are in an intermediate situation where the inputs are sent to the worker through a not-so-secure network where encryption is not a luxury. For this reason a CCE scheme comes in handy.

We note that, when encryption is used instead of a secure channel, we must make sure that the adversary cannot submit inputs that he actually ignores by copying CCE ciphertexts. This can be prevented by using the non-malleability offered by sigma proofs to prevent any re-randomization of commitments, and by declaring duplicate commitments invalid (see [22] for details).

For readability, in the rest of the paper, we do not differentiate the commitments obtained through the DerivCom algorithm from other commitments produced in the proofs below, and, without loss of generality, we assume that the Com algorithm stands either for the contraction of DerivCom(Enc) when a CCE scheme is used either for some commitment scheme when a secure channel is set up for the inputs transmission. In both cases, we assume a homomorphic perfectly hiding and computationally binding commitment scheme.

3.2 Non Interactive Zero-knowledge Proof of Knowledge

The second tool that we need is non-interactive zero-knowledge proof of knowledge (NIZKPK). Below we explicit the different relations for the proofs that we use in our construction.

The first NIZKPK is the classical or-proof of knowledge [23] denoted $\pi_{\mathsf{or}}(d)$. Another kind of well-known NIZKPK is the proof of equality of discrete logarithms between two commitments [24] that we refer to as $\pi_{\mathsf{DL}}(d_1, d_2)$. The proof of the opening of the commitment is denoted $\pi_{\mathsf{ope}}(d)$. We also rely on proofs for multiplications $\pi_{\mathsf{mul}}(d_1, d_2, d_3)$, on range proofs $\pi_{\mathsf{ran}}(d, I)$ and on comparison proofs[1] $\pi_{<}(d_1, d_2, d_3)$ which are essentially range proofs. We give the relations for these proofs respectively:

[1] Since committed values belong to $\mathbb{Z}_q$, this comparison operator makes sense only on a small interval of $\mathbb{Z}_q$ where one can define a natural order. Typically an interval centered in $0 \in \mathbb{Z}_q$.

$R^{\mathsf{or}} := \{(d,(x,o))|\mathsf{Verify}(d,o,x) = 1 \wedge (x = 0 \vee x = 1)\}$
$R^{\mathsf{DL}} := \{((d_1,d_2),(x,o_1,o_2))|\mathsf{Verify}(d_1,o_1,x) = 1 \wedge \mathsf{Verify}(d_2,o_2,x) = 1\}$
$R^{\mathsf{ope}} := \{(d,(x,o))|\mathsf{Verify}(d,o,x) = 1\}$
$R^{\mathsf{mul}} := \{((d_1,d_2,d_3),(x_1,o_1,x_2,o_2,o_3))|x_3 = x_1x_2 \wedge_i \mathsf{Verify}(d_i,o_i,x_i) = 1\}$
$R^{\mathsf{ran}} := \{(d,(x,o))|\mathsf{Verify}(d,o,x) = 1 \wedge x \in I\}$
$R^{<} := \{((d_1,d_2,d_3),(x_1,o_1,x_2,o_2,x_3,o_3))|x_3 = x_1 < x_2 \wedge_i \mathsf{Verify}(d_i,o_i,x_i) = 1\}$

The details of the Σ-protocols for relations R^{mul}, R^{ran} and $R^{<}$ are given in Appendix C. Note that the soundness of the proof π_{mul} relies on the binding property of the commitment scheme since d_3 could theoretically open to any value. A direct way to obtain a range proof that $x \in [0, 2^{k+1}[$ is by composing k proofs $\pi_{\mathsf{or}}(b_i)$ where b_i is the binary decomposition of x.

3.3 Generic Construction of Π^f_{PPAT}

Commitment consistent encryption and NIZKPK are the building blocks of protocol Π^f_{PPAT}.

The protocol is fairly simple: each client computes a $c_i \leftarrow \mathsf{Enc}(x_i)$ of his private input. From c_i, C_i derives a perfectly hiding commitment $d_i \leftarrow \mathsf{DerivCom}(c_i)$ and, computes a $\pi_{\mathsf{ver}}(d_i)$, for example $\pi_{\mathsf{or}}(d_i)$, $\pi_{\mathsf{ran}}(d_i, I)$ or by default, $\pi_{\mathsf{ope}}(d_i)$. Then, C_i publishes d_i and $\pi_{\mathsf{ver}}(d_i)$ on **PB**, and sends c_i to $\mathcal{W}$. After having decrypted every c_i to get the clients' private inputs, $\mathcal{W}$ computes $y := f(x_1, \cdots, x_n)$. From the commitments published on **PB**, $\mathcal{W}$ computes $d_y := f(d_1, \cdots, d_n)$ as well as the NIZKPK-s for each operator of f (except for $+$ and $-$ which are natural operators in the commitment space): the set of the NIZKPK and all the intermediary commitments created for the needs of the proofs are executed in parallel to form π_{cor}. $\mathcal{W}$ publishes y and π_{cor} on **PB**. Finally, each C_i verifies the correctness of π_{cor} in regards to y and the reconstruction of d_y.

As we will see in the next section, this is not the most efficient way to achieve the perfectly private audit trail since there are lots of cases where the verification of the output of the function can be done without recomputing the entire function in the commitment space.

4 Applications

Until now, we have seen how to generate a perfectly private audit trail of computation from the blueprint of any function. As we will see through several examples, there is a more direct way to provide the π_{cor} that guarantees the correctness of the output. The main idea is that, once given the result of the function it is much simpler to verify that it is correct. For example, once you are told that 8128 is the square root of $66,064,384$, it costs you only one squaring to agree while finding the square root by hand calculus is trickier.

In the following applications, we show how to use this trick to reduce the complexity of the proof for the client compared with the original complexity of the algorithm computing the result as it must be done in classical SMC. We selected unrelated problems to illustrate the ease of application of our technique and we point out in Appendix G other examples that may turn into good candidates.

4.1 System of Linear Equations

The first application is solving a system of linear equations. It is involved as a subroutine in many algorithms as, among others, in the Lagrange polynomial interpolation or in linear programming techniques. In linear programming, the goal is to optimize a solution under a set of linear constraints. This kind of scenario fits very well in a multi-party setting. We illustrate it by an example in which a set of companies in a production line agree to cooperate in order to optimize the production of some goods but do not wish to divulge their internal work flow to each other. The gain for the companies is lower costs and the ability to reallocate resources. Nowadays, all the solutions impair the privacy in one way or another, thus preventing such benefits.

Consider the following system of linear equations L:

$$L \equiv \begin{cases} \alpha_{1,1}z_1 + \cdots + \alpha_{1,n}z_n & = b_1 \\ \cdots & \cdots \\ \alpha_{m,1}z_1 + \cdots + \alpha_{m,n}z_n & = b_m \end{cases} \Leftrightarrow AZ = B$$

where $A \in \mathbf{M}^{m\times n}$, $B \in \mathbf{M}^{m\times 1}$, $\mathbf{M}$ is the coefficient space, and Z is the matrix of variables of dimension $(n \times 1)$. The unique solution, if it exists, is $Z_s = A^{-1}B$. When the matrix is not invertible, one might produce Z_{nts} a non trivial solution of the homogeneous system $AZ = 0$.

In a multi-party setting, the constraints are given by the clients. Here we consider K clients where $K = mn$. The clients are indexed by $i = (1,1), \cdots, (m,n)$ as are the coefficients α_i of the matrix A. The simplest scenario is that α_i is the private input of client C_i while the independent coefficients b_j for $j = 1, \cdots, m$, are known to everyone.

Each C_i computes $c_i \leftarrow \mathsf{Enc}(\alpha_i)$ and derives from it a commitment on α_i: $d_i \leftarrow \mathsf{DerivCom}(c_i)$. The d_i-s are published on **PB** as well as a proof $\pi_{\mathsf{ope}}(d_i)$ computed by C_i. The encryptions c_i are passed on to $\mathcal{W}$. We can combine each d_i on **PB** to form the matrix D which can be seen as a commitment on A. After having decrypted each c_i to get α_i, $\mathcal{W}$ computes the inverse matrix A^{-1} with his favourite method and thus the solution $Z_s = A^{-1}B$. The worker then publishes Z_s on **PB** along with π_{cor}. This proof consists of a list of m openings $o_1, \cdots, o_m$ where o_j is the opening of $b'_j = d_{j,1}z_1 + \cdots + d_{j,n}z_n$. Indeed, to verify the result, each client computes $B' := DZ_s$ and checks that the opening of each entry of this $(m,1)$-matrix is valid and that B' opens on the values of B. In the case of a non trivial solution Z_{nts} occurring when A is not invertible, the worker opens $B' := DZ_{nts}$ which must be a series of commitments on zero.

One might also want to include B in the private inputs of the clients. In this case, the π_{cor} is a bit different. Instead of giving the openings o_j, $\mathcal{W}$ provides a $\pi_{\mathsf{DL}}(b'_j, b_j)$ that b'_j commits on the same value as b_j for $j = 1, \cdots, m$.

π_{cor} **is Zero-Knowledge:**

- *Completeness*: It is clear that, given $o_1, ..., o_m$, any client can verify that Z_s is indeed the solution of the linear system.
- *Soundness*: This relies on the computationally binding property of the commitment scheme.
- *Perfect ZK*: As the commitment scheme is perfectly hiding, the openings of the commitments must be uniformly distributed in the space of openings.

Complexity. For the client, the complexity cost is exactly linear in the number of clients. In fact, the complexity bottleneck of the protocol is how to find Z. Either by inverting A, by the Gauss-Jordan elimination or more efficiently by the LU decomposition method, computing Z_s requires about $O(n^3)$ operations(or $O(n^{2.373}$ with the best current algorithm). It is also noteworthy that these algorithms often require branching when for example, searching the pivot in a row. However, when performed in SMC, these operations may become costly.

4.2 Auctions

Another type of problem that benefits from our approach is electronic auctions. We consider a setting of simple auctions where n clients submit one bid each. The result of the auction consists of a list of the sorted bids. More precisely, each C_i computes $c_i \leftarrow \mathsf{Enc}(x_i)$ where the bid $x_i \in I = [0, L[$. From c_i, C_i derives $d_i \leftarrow \mathsf{DerivCom}(c_i)$ and computes $\pi_{\mathsf{ran}}(d_i, I)$. While c_i is sent to $\mathcal{W}$, each C_i publishes d_i and $\pi_{\mathsf{ran}}(d_i, I)$ on **PB**. $\mathcal{W}$ computes the sorted list $(x'_1, \cdots, x'_n)$ from $(x_1, \cdots, x_n)$ with his favourite algorithm. From the sorted list, $\mathcal{W}$ rearranges the d_i to produce a sorted list of commitments $d'_1, \cdots, d'_n$. Then, $\mathcal{W}$ computes $n-1$ commitments $e_1, \cdots, e_{n-1}$ where $e_i := d_i \geq d_{i+1}$ which requires $n-1$ proofs $\pi_{\mathsf{ran}}(d_i - d_{i+1},]-L, L[)$ denoted π_i. Thus, $\pi_{\mathsf{cor}} := ((e_1, o_1, \pi_1) \wedge \cdots \wedge (e_{n-1}, o_{n-1}, \pi_{n-1}))$ where o_i is an opening of r_i which must imply that r_i commits on 1. $\mathcal{W}$ publishes π_{cor} on **PB** along with $d'_1, \cdots, d'_n$. Then, each client verifies π_{cor} to validate the result of the auction.

Note that, while there is a strong guarantee for the client that the winner(s) of the auction are legitimate winner(s), the winning bid and every other bids remain perfectly private. However, if required, $\mathcal{W}$ could also have revealed the winning bids by publishing the openings of the commitments. It is possible to transform this protocol into a sorting protocol that does not reveal which client's bid arrives in which position. This can be done by, first, randomizing the commitments on **PB** and, then, providing a proof of shuffle that the sorted list of commitments comes from the randomized list. The work of Terelius and Wikström [25] proposes such an efficient technique that adapts our commitment consistent approach. In this way, one could use the protocol as a subroutine of an algorithm that needs sorting.

π_{cor} is **ZK**: This is clear since π_{cor} is formed by a series of NIZKPK.

Complexity. As in the case of the linear system, the complexity for the client is linear in the number of clients. For the worker though, the highest complexity comes from the sorting algorithm (for example $O(n\log(n))$ in the case of Quicksort). However, as it is done on clear values, the cost is marginal compared with the linear number of range proofs to compute.

4.3 Shortest Path

In this third example, we aim at showing that realizing more complex protocols can be done without much difficulty. In the case of the shortest path, we consider a directed graph with m vertices and n weighted edges. The goal is to find the shortest path from a source node to a target node which is the path that minimizes the sum of the weights. We denote by $v_1, \cdots, v_m$ the vertices, while $e_k^{i,j}$ is the edge from v_i to v_j, numbered k in the edges list. Similarly, we denote $w_k^{i,j}$ the positive weight of edge $e_k^{i,j}$.

This problem has been studied in SMC since it offers a potential solution to privacy-preserving GPS guidance in which the guided person does not want to reveal its location to the service provider. In a multi-party setting involving more than two players, one can imagine the following scenario. A set of concurrent delivering companies possessing connections with spare room available for goods might be appealed to work together to offer a joined transport solution to a client without disclosing private information.

The Bellman-Ford algorithm solves the shortest path problem in $O(mn)$ operations. This algorithm maintains two lists: a predecessors list $\boldsymbol{pred}$ and a distances list $\boldsymbol{dist}$. While the algorithm executes, $pred_i$ designates the predecessor vertex of v_i while $dist_i$ stores the distance of v_i which is the weight of the current shortest path from the source to v_i.

In a multi-party setting we assume that each client C_i has w_i as private input. It is possible to turn Bellman-Ford algorithm in its secure version using classical SMC or using our technique. As previously, the derived commitment $d_i \leftarrow \mathsf{DerivCom}(c_i)$ are published on **PB**, while $\mathcal{W}$ gathers the $c_i \leftarrow \mathsf{Enc}(w_i)$ of the clients private inputs. Then $\mathcal{W}$ decrypts and computes the shortest path. The algorithm requires a supremum value denoted $\top$ which is the maximum weight a path might have plus one. We define $\top$ as $n.L - n + 1$, where L is the bound on the weights of the edges: $w_i < L$. As a result, we also require C_i to publish with d_i, a $\pi_{\mathsf{ran}}(d_i, L)$ on **PB**. This proof must later be verified by the other clients and $\mathcal{W}$.

Let us now focus on the computation of π_{cor} by $\mathcal{W}$. This is done by computing Algorithm 1 on the commitments and by providing an NIZKPK when necessary.

The predecessor of a vertex is represented by a commitment to the number of the vertex (lines 2 and 8) as well as the commitments on 0 and $\top$ on lines 3, and 7. These commitments can be computed once and then reused when needed. Their openings should also be given in π_{cor}. The comparison on line 7 requires a π_{ran} and the two multiplications on lines 8 require each a π_{mul}. All these proofs are aggregated in π_{cor}.

Algorithm 1: Secure shortest path based on Bellman-Ford's algorithm

Input: A graph $G = (V, E)$ where V is the list of vertices $v_1, \cdots, v_m$ and E the list of edges $e_1, \cdots, e_n$ associated to a list of committed weights $d_1, \cdots, d_n$. One of the vertex is labelled *source*.

Output: The predecessors list ***pred*** and/or the total distances list ***dist***.

for $i \leftarrow 1$ **to** m **do**
  $pred_i \leftarrow \mathsf{Com}(i)$
  if $v_i = source$ **then**
    $dist_i \leftarrow \mathsf{Com}(0)$
  else
    $dist_i \leftarrow \mathsf{Com}(\top)$
  end
end
for $k \leftarrow 1$ **to** m **do**
  for $l \leftarrow 1$ **to** n **do**
    $e_l^{i,j} = e_l$
    $z \leftarrow dist_i - dist_j + d_l$
    $x \leftarrow z < \mathsf{Com}(0)$
    $dist_j \leftarrow dist_j + x \cdot z$
    $pred_j \leftarrow pred_j + x \cdot (\mathsf{Com}(i) - pred_j)$
  end
end

π_{cor} is **ZK**: This is clear by the generic construction of π_{cor} (see Section 3.3).

Complexity. The verification algorithm has exactly the same complexity as the algorithm itself since it is a secure version of it. As a result the complexity of the verification of the proof is $O(mn)$ for the client. Of course, depending on the context, various shortest path algorithms exists some of which are more efficient than Bellman-Ford. However, we ignore if a verification proof simpler in terms of complexity than the best computation algorithm exists. Nevertheless, this example shows that it is always possible to obtain a complexity equal to the complexity of the algorithm.

5 Prototype Implementation

A generic implementation of the protocol has been realized in Python. The main objective was to create a simple framework to emulate the clients-worker interaction and measure the load of work of each party in different scenarios. Our entire code is available at *https://github.com/mpfeppat/mpfeppat*. We also provide the code of the applications tested in this paper to facilitate the reproduction of our results. Appendix D points out precautions that must be taken prior to using an implementation of our technique. We indicate which part of the code on the client's and on the worker's side must be audited and how.

The implementation aims to be light and reasonably efficient by using optimized algorithms and techniques. Our implementation is a prototype. However

Table 1. complexity and size cost for primitives and NIZKPK.

Legend		Computation	Verification	Size
$\mathtt{A}$: EC Point addition in G_1	Enc	$8\mathtt{Sm_p} + 3\mathtt{A}$	\	$5\mathtt{U} + 3$
	Com	$2\mathtt{Sm_p} + 1\mathtt{A}$	$2\mathtt{Sm_p} + 1\mathtt{A}$	$\mathtt{U} + 1$
$\mathtt{Sm_p}, \mathtt{Sm}$: EC scalar multiplication in G_1 with an without precomputation	π_{ope}	$2\mathtt{Sm_p} + 1\mathtt{A}$	$\mathtt{Sm} + 2\mathtt{Sm_p} + 2\mathtt{A}$	$3\mathtt{U}$
	π_{or}	$4\mathtt{Sm_p} + 2\mathtt{A}$	$2\mathtt{Sm} + 3\mathtt{Sm_p} + 3\mathtt{A}$	$4\mathtt{U}$
	π_{DL}	$4\mathtt{Sm_p} + 2\mathtt{A}$	$2\mathtt{Sm} + 4\mathtt{Sm_p} + 4\mathtt{A}$	$4\mathtt{U}$
	π_{mul}	$6\mathtt{Sm_p} + 3\mathtt{A}$	$4\mathtt{Sm} + 5\mathtt{Sm_p} + 6\mathtt{A}$	$6\mathtt{U}$
$\mathtt{U}$: λ-bit long integer	$\pi_{\mathsf{ran}}(2^{k+1})$	$6k\mathtt{Sm_p} + 3k\mathtt{A}$	$(3k-1)\mathtt{Sm} + 3k\mathtt{Sm_p} + (4k-1)\mathtt{A}$	$5k\mathtt{U} + k$

it is already a good indicator of performance. Nevertheless, one might expect a nice efficiency gain when using optimized code in C for example. This should be worth for specifically designed applications. In this regard, we do not claim that we have the best known time results for our applications.

The CCE scheme of [22] is implemented through Elliptic Curves (EC). We denote $\mathbb{F}_p$ the prime field where p is a λ-bit prime. The Pedersen-like commitment part and the ElGamal-like encryption part of the CCE scheme are performed respectively in $G_1 := E(\mathbb{F}_p)$ and $G_2 \subset E'(\mathbb{F}_{p^2})$, two prime order q groups where $E(\mathbb{F}_p), E'(\mathbb{F}_{p^2})$ are elliptic curves. Complete details are provided in Appendix E and the complexity analysis of our three test applications is given in Appendix F.

In the following, we denote by $\mathtt{A}$ the EC Point addition in G_1 and by $\mathtt{Sm}$ and $\mathtt{Sm_p}$, the scalar multiplication of an EC Point in G_1 without and with precomputation respectively. We remark that one operation in G_2 costs approximately twice the same operation in G_1. Consequently one encryption costs $8\mathtt{Sm_p} + 3\mathtt{A}$. In Table 1, we list the cost of different tools we use in our algorithms. For each NIZKPK, we split the cost between the computation and the verification. The proofs are computed in parallel whenever possible which imply that the computational cost of π_{cor} is not always the sum of the cost of each intermediate proof but rather a fraction of this sum. In this table, we also indicate the size cost of each proof and the commitment. As a base unit, we write $\mathtt{U}$ for a λ-bit long integer which is the length of a random number in $\mathbb{F}_p$ or roughly in $\mathbb{Z}_q$. Storing one EC point could be done by storing its x-coordinate plus 1 bit of sign. Thus, storing one ciphertext necessitates 2 EC points in G_2 and 1 in G_1 which sum up to $5\mathtt{U} + 3$.

Several tests were performed on a standard laptop: Intel® Core i5-3320M CPU @ 2.60GHz×4 with 7.7 GB of RAM. For these tests, the security parameter λ is 256-bit long. Even though this is the current security requirement for EC based cryptosystems, we argue that we do not need such a high security parameter for our protocol. Indeed, a polynomial time adversary that would be able to break the correctness of the scheme needs to run an attack against the binding property of the commitment scheme (in our case, break the DDH problem). However, at the time scale of the protocol execution, this attack has to be performed in a short amount of time. For this reason, we suspect that a smaller

Table 2. Timings (in seconds) and proof size for the three applications – the proof sizes are computed for a security parameter λ of 256-bit long.

	number of clients $\lvert\mathcal{C}\rvert$	parameters	worker	client	proof size
Linear System Solving	16	square system of size $\sqrt{\lvert\mathcal{C}\rvert} \times \sqrt{\lvert\mathcal{C}\rvert}$	$1.86e10^{-1}$	$4.14e10^{-2}$	384B
	256		3.03	$5.62e10^{-1}$	1.54KB
	4096		52.34	8.8	6.14KB
Auctions	10		$3.94e10^{-1}$	$3.87e10^{-1}$	22.79KB
	100		4	4.17	250.5KB
	1000		40.08	42.04	2.53MB
Shortest Path	4	number of vertices = number of edges = $\lvert\mathcal{C}\rvert$	$2.57e10^{-1}$	$4.81e10^{-1}$	54.81KB
	16		2.57	6.85	864.7KB
	64		35.03	105.7	13.79MB

security parameter would still allow a high level of security and decrease the computational burden of the participants.

The time results including all the computations are presented in Table 2. The results show realistic time performance. Solving a linear system of equations for 4096 clients (square matrix of size 64) takes 9 seconds of verification per client. In our auctions application between 100 bidders, each client needs about 4 seconds to verify the proof π_{cor}. Finally, verifying that a path is indeed the shortest path in a graph with 16 vertexes takes about 7 seconds per client. In theses two last applications, the time needed for the worker to decrypt and prepare the proof π_{cor} is roughly the same as the client. For the linear system, however, the time needed for the worker is around 52 seconds, partly due to the large number of clients.

The results show clearly the linearity in the number of clients in the first two applications while the complexity of the shortest path follows the quadratic complexity of the algorithm. The main limitations of efficiency might come from our use of Python and the generic Gmpy package for basic modular operations of addition and multiplication as these do not provide algebraic computations optimized for finite fields of a given characteristic. As a point of comparison, the auctions protocol of Rabin et al. [14] based on symmetric cryptography and cut-and-choose proofs achieve a 100 bids auctions with a security parameter of size 40. The worker needs 4.11 minutes to prepare the proof and the clients, less than one minute to verify it while the proof in itself weight for 1.45GB.

6 Conclusion

Current progress and real world applications in the field of secure multi-party computations and multi-party verifiable computation are positive indicators for this branch of cryptography. Faster and reliable but also user-friendly solutions are provided to meet the needs of an emerging sector of activity.

This work aims at proposing a simple and efficient solution to evaluate multi-party function in a clients-worker setting where the clients want a strong guar-

antee over the correctness of the result. Our solution is based on perfectly hiding commitments posted on a public bulletin board for which a worker will be bound to and will provide a computationally sound proof of correctness. We combine this commitment with an encryption in a primitive called *commitment consistent encryption* to provide a generic and easy-to-set up protocol that is secure against passive adversary for the privacy and secure against active adversary for the correctness of the function evaluation. As a result, our protocol provides a *perfectly private audit trail*.

Moreover, we show that this setting allows the clients to gain in complexity for the verification of the proof when this verification is cheaper than the algorithm used to compute the result. As a side effect, the worker is able to use his own algorithm to compute the result of the function without disclosing the intellectual property of his algorithm. This is of particular interest when the worker is a company specialized in algorithmic optimization. We illustrate the ease of use of our technique by three – rather simple but already used in real world – applications. We also provide timing results measured on our prototype implementation that indicate efficiency even though we point out that improvements could be achieved with clever optimizations.

Acknowledgements. This work has been funded by the Brussels Region INNOVIRIS project SeCloud. Part of this work was done while Édouard Cuvelier was funded by a FRIA grant of the F.R.S.-FNRS. The authors are grateful to the anonymous reviewers for their constructive feedback. They also like to thank Sylvie Baudine for her help in improving the paper.

References

1. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majorit – or: Breaking the SPDZ limits. In: Computer Security – ESORICS 2013. Springer (2013) 1–18
2. Baum, C., Damgård, I., Orlandi, C.: Publicly auditable secure multi-party computation. IACR Cryptology ePrint Archive **2014** (2014) 75
3. Aly, A., Cuvelier, E., Mawet, S., Pereira, O., Vanvyve, M.: Securly Solving Simple Combinatorial Graph Problems. In: Financial Cryptography and Data Security 2013. Volume 7859 of Lecture Notes in Computer Science. (4 2013) 239–257
4. Bogetoft, P., Christensen, D.L., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J.D., Nielsen, J.B., Nielsen, K., et al.: Secure multiparty computation goes live. In: Financial Cryptography and Data Security. Springer (2009) 325–343
5. Adida, B., De Marneffe, O., Pereira, O., Quisquater, J.J.: Electing a university president using open-audit voting: Analysis of real-world use of Helios. EVT/WOTE **9** (2009) 10–10
6. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: IEEE Symposium on Security and Privacy (SP), IEEE (2013) 238–252
7. Costello, C., Fournet, C., Howell, J., Kohlweiss, M., Kreuter, B., Naehrig, M., Parno, B., Zahur, S.: Geppetto: Versatile verifiable computation. In: Proceedings of the IEEE Symposium on Security and Privacy, IEEE (May 2015)

8. Backes, M., Barbosa, M., Fiore, D., Reischuk, R.M.: Adsnark: Nearly practical and privacy-preserving proofs on authenticated data. Cryptology ePrint Archive, Report 2014/617 (2014) http://eprint.iacr.org/.
9. Zhang, Y., Papamanthou, C., Katz, J.: Alitheia: Towards practical verifiable graph processing. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM (2014) 856–867
10. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Advances in Cryptology – CRYPTO 2010. Springer (2010) 465–482
11. Choi, S.G., Katz, J., Kumaresan, R., Cid, C.: Multi-client non-interactive verifiable computation. In: Theory of Cryptography. Springer (2013) 499–518
12. Goldwasser, S., Gordon, S.D., Goyal, V., Jain, A., Katz, J., Liu, F.H., Sahai, A., Shi, E., Zhou, H.S.: Multi-input functional encryption. In: Advances in Cryptology – EUROCRYPT 2014. Springer (2014) 578–602
13. Gordon, S.D., Katz, J., Liu, F.H., Shi, E., Zhou, H.S.: Multi-client verifiable computation with stronger security guarantees. In: Theory of Cryptography. Springer (2015) 144–168
14. Rabin, M.O., Servedio, R.A., Thorpe, C.: Highly efficient secrecy-preserving proofs of correctness of computation (April 18 2008) US Patent App. 12/105,508.
15. Parkes, D.C., Rabin, M.O., Shieber, S.M., Thorpe, C.: Practical secrecy-preserving, verifiably correct and trustworthy auctions. Electronic Commerce Research and Applications **7**(3) (2008) 294–312
16. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: IEEE 54th Annual Symposium on Foundations of Computer Science, IEEE Computer Society (2001) 136–136
17. Dolev, D., Dwork, C., Naor, M.: Non-malleable cryptography. In: SIAM Journal on Computing. (1998) 542–552
18. Bernhard, D., Pereira, O., Warinschi, B.: How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In: ASIACRYPT 2012. Number 7658 in LNCS, Springer (12 2012) 626–643
19. Damgård, I.: On Σ-protocols. http://www.daimi.au.dk/ ivan/Sigma.ps (2004)
20. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Advances in Cryptology – CRYPTO86, Springer (1987) 186–194
21. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the 1st ACM conference on Computer and communications security, ACM (1993) 62–73
22. Cuvelier, E., Pereira, O., Peters, T.: Election verifiability or ballot privacy: Do we need to choose? In: Computer Security–ESORICS 2013. Springer (2013) 481–498
23. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Advances in Cryptology – CRYPTO94, Springer (1994) 174–187
24. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Advances in Cryptology – CRYPTO92, Springer (1993) 89–105
25. Terelius, B., Wikström, D.: Proofs of restricted shuffles. In: Progress in Cryptology – AFRICACRYPT 2010. Springer (2010) 100–113
26. Damgård, I., Fujisaki, E.: A statistically-hiding integer commitment scheme based on groups with hidden order. In: Advances in Cryptology – ASIACRYPT 2002. Springer (2002) 125–142

27. Canard, S., Coisel, I., Jambert, A., Traoré, J.: New results for the practical use of range proofs. In: Public Key Infrastructures, Services and Applications. Springer (2014) 47–64
28. Pereira, G.C., Simplício Jr, M.A., Naehrig, M., Barreto, P.S.: A family of implementation-friendly BN elliptic curves. Journal of Systems and Software **84**(8) (2011) 1319–1326
29. Barreto, P.S., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Selected areas in cryptography, Springer (2006) 319–331
30. Haustenne, L., De Neyer, Q., Pereira, O.: Elliptic curve cryptography in javascript. IACR Cryptology ePrint Archive **2011** (2011) 654
31. Joye, M.: Fast point multiplication on elliptic curves without precomputation. In: Arithmetic of Finite Fields. Springer (2008) 36–46
32. Fan, J., Vercauteren, F., Verbauwhede, I.: Faster $\mathbb{F}_p$-arithmetic for cryptographic pairings on barreto-naehrig curves. In: Cryptographic Hardware and Embedded Systems-CHES 2009. Springer (2009) 240–253
33. Devegili, A.J., Scott, M., Dahab, R.: Implementing cryptographic pairings over barreto-naehrig curves. In: Pairing-Based Cryptography–Pairing 2007. Springer (2007) 197–207
34. Beuchat, J.L., González-Díaz, J.E., Mitsunari, S., Okamoto, E., Rodríguez-Henríquez, F., Teruya, T.: High-speed software implementation of the optimal ate pairing over barreto–naehrig curves. In: Pairing-Based Cryptography-Pairing 2010. Springer (2010) 21–39
35. Aranha, D.F., Karabina, K., Longa, P., Gebotys, C.H., López, J.: Faster explicit formulas for computing pairings over ordinary curves. In: Advances in Cryptology–EUROCRYPT 2011. Springer (2011) 48–68
36. Scott, M., Benger, N., Charlemagne, M., Perez, L.J.D., Kachisa, E.J.: On the final exponentiation for calculating pairings on ordinary elliptic curves. In: Pairing-Based Cryptography–Pairing 2009. Springer (2009) 78–88

A Additional functionalities

We might change Protocol Π^f_{PPAT} slightly to provide another functionality where the evaluation y of the function is not disclosed to the clients in step 3. However, the clients receive a commitment d_y on the output y with the guarantee that $y = f(x_1, ..., x_n)$. The worker proves that he performed the computations honestly but without revealing the result. In this way, we might use the protocol as a subroutine to securely compute larger functions.

It is also possible to modify the protocol to achieve security against an active adversary $\mathcal{A}_{a^*}$ that would dynamically modify the inputs of the corrupted clients. In this case, we require that the clients do not send directly their private inputs and openings x_i, o_i to $\mathcal{W}$ in step 1. Instead the clients wait until every other client C_i has published a commitment d_i and a proof $\pi_{\mathsf{ver}}(d_i)$ on **PB**. At this point only, they send their x_i, o_i to $\mathcal{W}$ through the secure channel. This additional step prevents the adversary from choosing the inputs of the corrupted clients when they receive the inputs of the honest clients. While this change looks quite simple to stop these kinds of attacks, its major drawback is that it adds an extra step for the clients during which they have to check that the commitments have

all been published on **PB** to proceed with the rest of the protocol. This reduces the benefits of the non-interactivity of the original protocol one may seek. Since considering the realization of the (modified) functionality $\mathcal{F}^f_{\mathcal{A}_{a^*}}$ would not bring tremendous changes to discuss, we now focus only on the realization of $\mathcal{F}^f_{\mathcal{A}_a}$ by protocol Π^f_{PPAT}.

B Proofs of Security

B.1 Proof of Theorem 1

Proof (Complete proof). To prove the security of the scheme, we show that the view of any environment $\mathcal{E}$ is identical whether it is a real execution of the protocol Π^f_{PPAT} under the presence of adversary $\mathcal{A}_p$ or an ideal execution emulating the functionality $\mathcal{F}^f_{\mathcal{A}_p}$ in the presence of an adversary $\mathcal{S}$ simulating the execution of $\mathcal{A}_p$.

We create a set of games $\mathcal{G}_i$ that are indistinguishable each from the previous or the next one. The first game $\mathcal{G}_1$ representing the real execution of the protocol and the last game $\mathcal{G}_5$ being the ideal execution emulating the functionality $\mathcal{F}^f_{\mathcal{A}_p}$.

Let $\mathsf{exec}^{\mathcal{G}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x})$ be the random variable describing the output of the environment $\mathcal{E}$ when interacting with adversary $\mathcal{A}$ and the parties in the game $\mathcal{G}$ on input $\boldsymbol{x}$. We will prove that, for any environment $\mathcal{E}$, there exists a simulator $\mathcal{S}$ such that $\mathsf{exec}^{\mathcal{G}_1}_{\mathcal{A}_p,\mathcal{E}}(\boldsymbol{x}) = \mathsf{exec}^{\mathcal{G}_5}_{\mathcal{S},\mathcal{E}}(\boldsymbol{x})$.

Without loss of generality, and for the sake of the proof, we rearrange the indexes of the clients by partitioning the set of n clients into two subsets: $\{1,...,n\} = \mathcal{HON} \cup \mathcal{COR}$ where $\mathcal{HON} = \{1,...,k\}$ indexes honest clients while $\mathcal{COR} := \{k+1,...,n\}$ indexes corrupted clients.

$\mathcal{G}_2$ Starting from $\mathcal{G}_1$ we define a set of hybrid games denoted $\mathcal{G}_{2,i}$ for $i = 1,...,k$. In game $\mathcal{G}_{2,1}$, the proof published on **PB** by C_1 is replaced by a simulated proof. In game $\mathcal{G}_{2,i}$, the proofs published on **PB** by the C_j for $j \leq i$ are replaced by simulated proofs. At the end, we obtain game $\mathcal{G}_2 := \mathcal{G}_{2,k}$ where all the honest clients' proofs have been replaced by simulated ones. More precisely, the game $\mathcal{G}_{2,i}$ takes place as follows:

- for $j \in \mathcal{HON}$, when client C_j prepares $d_j, \pi_{\mathsf{ver}}(d_j)$ to be published on **PB**, compute a simulated proof $\pi^*_{\mathsf{ver}}(d_j)$ generated by the NIZKPK simulator $\mathcal{M}_{\mathsf{ver}}$ on inputs $(d_j, \mathfrak{e})$ where $\mathfrak{e}$ is a randomly generated challenge. If $j > i$, publish $(d_j, \pi_{\mathsf{ver}}(d_j))$ on **PB**. Otherwise, if $j \leq i$, publish $(d_j, \pi^*_{\mathsf{ver}}(d_j))$ on **PB**. Meanwhile, C_j sends his private input x_j as well as the opening of d_j to $\mathcal{W}$.
- for $j \in \mathcal{COR}$, when client C_j sends his inputs to $\mathcal{W}$ and publish on **PB**, proceed as in the protocol Π^f_{PPAT}.
- when all inputs values have been submitted, and if all proofs π_{ver} are valid, the worker proceeds as in the protocol Π^f_{PPAT} by computing and publishing $y := f(x_1,...,x_n)$ and π_{cor}.

$\mathcal{G}_1 \to \mathcal{G}_2$**:** We prove that for $i = 0, ..., k-1$, $\mathsf{exec}^{\mathcal{G}_{2,i}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x}) = \mathsf{exec}^{\mathcal{G}_{2,i+1}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x})$ where we define $\mathcal{G}_{2,0}$ as $\mathcal{G}_1$. We proceed by induction on i.
The first step is to prove identical views between $\mathcal{G}_{2,0}$ and $\mathcal{G}_{2,1}$. From any adversary $\mathcal{A}$ and environment $\mathcal{E}$ able to distinguish between $\mathcal{G}_{2,0}$ and $\mathcal{G}_{2,1}$, we build an adversary $\mathcal{B}$ against the *perfect zero-knowledge property* of the NIZKPK of the scheme. We claim that the advantage of $\mathcal{A}$ is at most the advantage of $\mathcal{B}$. Adversary $\mathcal{B}$ runs $\mathcal{A}$ and $\mathcal{E}$ internally and sets up a distinguisher $\mathcal{D}$. Instead of one bulletin board, $\mathcal{B}$ sets up two bulletin boards $\mathbf{PB}_0$ and $\mathbf{PB}_1$, then $\mathcal{B}$ flips a coin β. During the game, $\mathcal{E}$ and $\mathcal{A}$ only see $\mathbf{PB}_\beta$ as $\mathbf{PB}$. Then, $\mathcal{B}$ proceeds as follows:
- when client C_1 generates $x_1, o_1, d_1, \pi_{\mathsf{ver}}(d_1)$, $\mathcal{B}$ generates a simulated proof $\pi^*_{\mathsf{ver}}(d_1)$ with the NIZKPK simulator $\mathcal{M}_{\mathsf{ver}}$. Then, $(d_1, \pi_{\mathsf{ver}}(d_1))$ is published on $\mathbf{PB}_0$ while $(d_1, \pi^*_{\mathsf{ver}}(d_1))$ is published on $\mathbf{PB}_1$.
- when client C_j for $j \neq 1$ sends $d_j, \pi_{\mathsf{ver}}(d_j)$ to the board, $\mathcal{B}$ publishes $d_j, \pi_{\mathsf{ver}}(d_j)$ on $\mathbf{PB}_0$ and $\mathbf{PB}_1$.
- once all inputs have been submitted to $\mathcal{W}$, and if all proofs π_{ver} are valid, the worker prepares y and π_{cor} that are published on $\mathbf{PB}_0$ and $\mathbf{PB}_1$.

When $\mathcal{E}$ stops, $\mathcal{B}$ runs $\mathcal{D}$ on the local output of $\mathcal{E}$ and outputs whatever $\mathcal{D}$ outputs. When $\mathcal{E}$ and $\mathcal{A}$ have access to $\mathbf{PB}_0$, they are in $\mathcal{G}_{2,0}$ and when they have access to $\mathbf{PB}_1$, they are in $\mathcal{G}_{2,1}$. It follows that whenever $\mathcal{D}$ successfully distinguishes between the outputs in the two games, then $\mathcal{B}$ has a way to distinguish between the proofs $\pi_{\mathsf{ver}}(d_1)$ and $\pi^*_{\mathsf{ver}}(d_1)$. This contradicts the perfect zero-knowledge property of the NIZKPK.
Fixing some $1 < i_1 < k$, we assume that for all $1 \leq i < i_1$, we have $\mathsf{exec}^{\mathcal{G}_{2,i}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x}) = \mathsf{exec}^{\mathcal{G}_{2,i+1}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x})$. Then, we show that $\mathsf{exec}^{\mathcal{G}_{2,i_1}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x}) = \mathsf{exec}^{\mathcal{G}_{2,i_1+1}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x})$. This last equality is verified thanks to the same kind of reduction appearing in the first step of the induction. This concludes the proof by induction.

$\boxed{\mathcal{G}_3}$ We proceed similarly as in $\mathcal{G}_2$ and create a set of intermediary games $\mathcal{G}_{3,i}$ for $i = 0, ...k$ where $\mathcal{G}_{3,0}$ is defined as $\mathcal{G}_2$ and $\mathcal{G}_3 := \mathcal{G}_{3,k}$. In game $\mathcal{G}_{3,i}$, the commitments generated by the honest clients C_j for $j \leq i$ are replaced by freshly generated random commitments. In game $\mathcal{G}_3$, every commitment from the honest clients will be replaced by randomly distributed commitments. Game $\mathcal{G}_{3,i}$ for $i = 1, ..., k$ takes place as follows:
- for $j \in \mathcal{HON}$, when client C_j prepares $d_j, \pi_{\mathsf{ver}}(d_j)$ to be published on $\mathbf{PB}$, compute a commitment d^*_j and opening o^*_j on a random value x^*_j as well as two simulated proofs $\pi^*_{\mathsf{ver}}(d_j)$ and $\pi^*_{\mathsf{ver}}(d^*_j)$ respectively generated by the NIZKPK simulator $\mathcal{M}_{\mathsf{ver}}$ on inputs $(d_j, \mathfrak{e})$ and $(d^*_j, \mathfrak{e})$ respectively where $\mathfrak{e}$ is a randomly generated challenge. If $j > i$, publish $(d_j, \pi^*_{\mathsf{ver}}(d_j))$ on $\mathbf{PB}$. Otherwise, if $j \leq i$, publish $(d^*_j, \pi^*_{\mathsf{ver}}(d^*_j))$ on $\mathbf{PB}$. Meanwhile, if $j > i$, send x_j, o_j to $\mathcal{W}$. Otherwise if $j \leq i$, send x^*_j, o^*_j to $\mathcal{W}$.
- for $j \in \mathcal{COR}$, when client C_j sends his inputs to $\mathcal{W}$ and publishes on $\mathbf{PB}$, proceed as in the protocol Π^f_{PPAT}.
- when all inputs values have been submitted, and if all proofs π_{ver} are valid, $\mathcal{W}$ computes and publishes $y = f(x^*_1, ..., x^*_i, x_{i+1}, ..., x_n)$ and π_{cor} on $\mathbf{PB}$.

$\mathcal{G}_2 \rightarrow \mathcal{G}_3$**:** As in the previous game, we proceed by induction on $i = 0, ..., k-1$ to prove that $\mathsf{exec}^{\mathcal{G}_{3,0}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x}) = \mathsf{exec}^{\mathcal{G}_{3,k}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x})$. We begin by showing that $\mathsf{exec}^{\mathcal{G}_{3,0}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x}) = \mathsf{exec}^{\mathcal{G}_{3,1}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x})$. From any adversary $\mathcal{A}$ and environment $\mathcal{E}$ able to distinguish between $\mathcal{G}_{3,0}$ and $\mathcal{G}_{3,1}$, we build an adversary $\mathcal{B}$ against the *perfectly hiding property* of the commitment scheme. We claim that the advantage of $\mathcal{A}$ is at most the advantage of $\mathcal{B}$. Adversary $\mathcal{B}$ runs $\mathcal{A}$ and $\mathcal{E}$ internally and sets up a distinguisher $\mathcal{D}$. Instead of one bulletin board, $\mathcal{B}$ sets up two bulletin boards $\mathbf{PB}_0$ and $\mathbf{PB}_1$, then $\mathcal{B}$ flips a coin β. During the game, $\mathcal{E}$ and $\mathcal{A}$ only see $\mathbf{PB}_\beta$ as $\mathbf{PB}$. Then, $\mathcal{B}$ proceeds as follows:

- when client C_1 generates $x_1, o_1, d_1, \pi_{\mathsf{ver}}(d_1)$, $\mathcal{B}$ generates a commitment d_1^* and opening o_1^* on a random value x_1^* as well as two simulated proofs $\pi^*_{\mathsf{ver}}(d_1)$ and $\pi^*_{\mathsf{ver}}(d_1^*)$ with the NIZKPK simulator $\mathcal{M}_{\mathsf{ver}}$. Then, $(d_1, \pi^*_{\mathsf{ver}}(d_1))$ is published on $\mathbf{PB}_0$ while $(d_1^*, \pi^*_{\mathsf{ver}}(d_1^*))$ is published on $\mathbf{PB}_1$. Meanwhile, if $\beta = 0$, $\mathcal{W}$ receives x_1, o_1, else if $\beta = 1$, $\mathcal{W}$ receives x_1^*, o_1^*.
- for $j \in \mathcal{HON}, j \neq 1$, when client C_j generates $x_j, o_j, d_j, \pi_{\mathsf{ver}}(d_j)$, $\mathcal{B}$ generates a simulated proof $\pi^*_{\mathsf{ver}}(d_j)$ thanks to the NIZKPK simulator $\mathcal{M}_{\mathsf{ver}}$. Then, $(d_j, \pi^*_{\mathsf{ver}}(d_j))$ is published on $\mathbf{PB}_0$ and on $\mathbf{PB}_1$. Meanwhile, x_j, o_j are sent to $\mathcal{W}$.
- for $j \in \mathcal{COR}$, when client C_j generates $x_j, o_j, d_j, \pi_{\mathsf{ver}}(d_j)$, $\mathcal{B}$ proceeds as in game $\mathcal{G}_2$ and $d_j, \pi_{\mathsf{ver}}(d_j)$ is published on $\mathbf{PB}_0$ and $\mathbf{PB}_1$.
- once all inputs have been submitted to $\mathcal{W}$, and if all proofs π_{ver} are valid, the worker prepares y and π_{cor} to be published on $\mathbf{PB}_0$ and $\mathbf{PB}_1$.

When $\mathcal{E}$ stops, $\mathcal{B}$ runs $\mathcal{D}$ on the local output of $\mathcal{E}$ and outputs whatever $\mathcal{D}$ outputs. When $\mathcal{E}$ and $\mathcal{A}$ have access to $\mathbf{PB}_0$, they are in $\mathcal{G}_{3,0}$ and when they have access to $\mathbf{PB}_1$, they are in $\mathcal{G}_{3,1}$. The output of the distinguisher $\mathcal{D}$ is released by $\mathcal{B}$ and it comes that the success of $\mathcal{D}$ at differentiating between $\mathcal{G}_{3,0}$ and $\mathcal{G}_{3,1}$ is used by $\mathcal{B}$ to differentiate commitment d_1 from commitment d_1^*. This contradicts the perfectly hiding property of the commitment scheme. As before, if we assume that for a fixed $1 < i_1 < k$, we have for all $1 \leq i < i_1$ that $\mathsf{exec}^{\mathcal{G}_{3,i}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x}) = \mathsf{exec}^{\mathcal{G}_{3,i+1}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x})$, we can prove that $\mathsf{exec}^{\mathcal{G}_{3,i_1}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x}) = \mathsf{exec}^{\mathcal{G}_{3,i_1+1}}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x})$ by using the same kind of reduction as in the first step. Hence, we have showed that $\mathsf{exec}^{\mathcal{G}_2}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x}) = \mathsf{exec}^{\mathcal{G}_3}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x})$.

$\boxed{\mathcal{G}_4}$ In this game, we replace the proof π_{cor} generated by $\mathcal{W}$ by a simulated proof π^*_{cor} that is obtained through the NIZKPK simulator $\mathcal{M}_{\mathsf{cor}}$ on inputs y (output by $\mathcal{W}$) and the commitments presents on $\mathbf{PB}$.

$\mathcal{G}_3 \rightarrow \mathcal{G}_4$**:** From any adversary $\mathcal{A}$ and environment $\mathcal{E}$ able to distinguish between $\mathcal{G}_3$ and $\mathcal{G}_4$, we build an adversary $\mathcal{B}$ against the perfect zero-knowledge property of the NIZKPK of the scheme. The reduction works as the reductions in the proof that $\mathsf{exec}^{\mathcal{G}_1}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x}) = \mathsf{exec}^{\mathcal{G}_2}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x})$ yielding an adversary $\mathcal{B}$ able to break the *perfect zero-knowledge property* of the NIZKPK of the scheme.

$\boxed{\mathcal{G}_5}$ In the last game, we build the simulator $\mathcal{S}$ that proceeds as follows:

- When notified by the functionality $\mathcal{F}^f_{\mathcal{A}_p}$ of an input from an honest client C_i, $\mathcal{S}$ generates a commitment d_i^* and opening o_i^* on a random value x_i^*,

and computes a simulated proof $\pi^*_{\mathsf{ver}}(d_i^*)$ with the NIZKPK simulator $\mathcal{M}_{\mathsf{ver}}$. Then $\mathcal{S}$ publishes d_i^* and $\pi^*_{\mathsf{ver}}(d_i^*)$ on **PB**.
- When a corrupted client C_j submits a commitment and a proof to the board, and the opening of the commitment to the worker, $\mathcal{S}$ submits the opened input x_j to $\mathcal{F}^f_{\mathcal{A}_p}$ on behalf of C_j.
- When all input values have been submitted, and if all π_{ver} proofs are valid, $\mathcal{S}$ submits Compute to $\mathcal{F}^f_{\mathcal{A}_p}$ and obtains y in return. $\mathcal{S}$ uses the NIZKPK simulator $\mathcal{M}_{\mathsf{cor}}$ to obtain a simulated proof π^*_{cor} based on the commitments on the board and y. Then $\mathcal{S}$ publishes y and π^*_{cor} on the board.

$\mathcal{G}_4 \to \mathcal{G}_5$**:** This does not change the view of the environment, thus we have that $\mathsf{exec}^{\mathcal{G}_4}_{\mathcal{A},\mathcal{E}}(\boldsymbol{x}) = \mathsf{exec}^{\mathcal{G}_5}_{\mathcal{S},\mathcal{E}}(\boldsymbol{x})$.

B.2 Proof of Theorem 2

Proof (Complete proof). By contrast from $\mathcal{F}^f_{\mathcal{A}_p}$ of Theorem 1, we observe that $\mathcal{F}^f_{\mathcal{A}_a}$ offers correctness guarantees, but no privacy at all: all inputs are immediately given to the simulator.

Formally, we prove that there exists a simulator $\mathcal{S}$ such that the view for any environment $\mathcal{E}$ of an execution in the real world of the protocol Π^f_{PPAT} under the presence of an active adversary $\mathcal{A}_a$ is indistinguishable from the view of $\mathcal{E}$ of an execution in the ideal world of the protocol simulated by $\mathcal{S}$ and interacting with the functionality $\mathcal{F}^f_{\mathcal{A}_a}$. In other words, we show that $\mathsf{exec}^{\mathsf{REAL}}_{\mathcal{A}_a,\mathcal{E}}(\boldsymbol{x}) \approx \mathsf{exec}^{\mathsf{IDEAL}}_{\mathcal{S},\mathcal{E}}(\boldsymbol{x})$ where $\boldsymbol{x}$ is the input of the parties.

As previously, and without loss of generality, we partition and re-index the set $\{1, ..., n\}$ of the clients into the subsets $\mathcal{HON} \cup \mathcal{COR}$ where $\mathcal{HON} := \{1, ..., k\}$ is the set of honest clients and $\mathcal{COR} := \{k+1, ..., n\}$ is the set of corrupted clients.

In the ideal world, we describe how the simulator $\mathcal{S}$ interacts with $\mathcal{F}^f_{\mathcal{A}_a}$ in order to emulate the ideal execution of the protocol. $\mathcal{S}$ internally runs the corrupted worker and clients, relays their communications with the environment and with the bulletin board which is also emulated by the simulator. The simulator proceeds as follows:

- When notified by the functionality of an input x_i from an honest client C_i, $\mathcal{S}$ follows protocol Π^f_{PPAT} by generating and posting on the board a commitment d_i and a proof $\pi_{\mathsf{ver}}(d_i)$. Then, $\mathcal{S}$ submits (x_i, o_i) to the corrupted worker on behalf of C_i.
- $\mathcal{S}$ does not interfere when a corrupted client C_j submits a commitment and a proof to the board.
- When the corrupted worker submits the result y and the associated proof π_{cor}, $\mathcal{S}$ verifies each of the proofs on behalf of the honest clients and,
  - ▶ if they do not check, $\mathcal{S}$ does nothing and the protocol aborts as in the real world.

- otherwise, $\mathcal{S}$ runs the extractor E of the special soundness property of the proofs $\pi_{\mathsf{ver}}(d_i)$ posted on **PB** by client C_i for $i \in \mathcal{HON} \cup \mathcal{COR}$. By rewinding the protocol, $\mathcal{S}$ obtains two different transcripts $\mathfrak{t}, \mathfrak{t}'$ of the proof $\pi_{\mathsf{ver}}(d_i)$. With those inputs, the extractor E computes the witnesses (x'_i, o'_i) for each $(d_i, \pi_{\mathsf{ver}}(d_i))$.
  Then, the simulator checks that those values match the inputs of the honest clients and are consistent with the commitment posted by the corrupted clients. Also, $\mathcal{S}$ verifies if the result of the function posted by the corrupted worker is consistent with the result y' computed from the extracted inputs.
  If these verifications check, $\mathcal{S}$ relays the extracted inputs of the corrupted clients and submits Compute to the functionality. The functionality then sends its output to the clients.
  In the event that these verifications do not check, $\mathcal{S}$ does nothing and the protocol aborts. This happens in the four different cases[2]:
  (1) the extracted inputs from the honest clients C_i for $i \in \mathcal{HON}$, do not match those received by the functionality.
  (2) the extracted inputs from the corrupted clients C_j for $j \in \mathcal{COR}$, are not valid with respect to the proofs $\pi_{\mathsf{ver}}(d_j)$.
  (3) the result of the function is not valid with respect to the proof of correctness of the result π_{cor}.
  (4) we are not in cases 2 or 3 but the result of the function y^* provided by $\mathcal{W}$ is not equal to the result y computed by $\mathcal{S}$ from the extracted inputs.

We show that the view of the environment in the ideal world differs from its view in the real world only with negligible probability. Indeed the differences only occur in the four cases described above. However, given the assumptions made on the commitment scheme and the properties of the sigma proofs, respectively for each of the above cases, we have that:

(1) at least one commitment d_i possesses two different openings. This contradicts the *computationally binding property* of the commitment scheme.
(2) this contradicts the *soundness property* of the proof π_{ver}.
(3) this contradicts the *soundness property* of the proof π_{cor}.
(4) this again contradicts the *computationally binding property* of the commitment scheme. Indeed, if the corrupted worker outputs $y^* \neq y'$, it means that a least one of the inputs used by $\mathcal{W}$ to compute y^* is different from its counterpart extracted by E. Let us name this input x_l^* for some $l \in \mathcal{COR}$ $(x'_l \neq x_l^*)$. Now, we can use $\mathcal{A}_a$ and the soundness extractor E to produce two different openings for the commitments d_l. In particular, we have (x'_l, o'_l), the extracted opening and (x_l^*, o_l^*), the opening used by $\mathcal{W}$ in the computation of y^*.

As the probability of each of theses cases occurring is negligible, thereby we conclude the proof.

[2] plus any meaningful combination of them

C Non Interactive Zero-Knowledge Proofs of Knowledge

In the following, we consider traditional Pedersen commitment which is a perfectly hiding and computationally binding commitment. We have $d = g^x h^r \leftarrow \mathsf{Com}(x)$ for generators g and h such that $\langle g \rangle = \langle h \rangle = G$, a cyclic group of prime order and that $\log_h g$ is unknown. This commitment is essentially the same commitment output by the DerivCom algorithm of our concrete implementation as detailed in Appendix E. Of course, we could have chosen other commitment scheme with the right properties as well.

C.1 NIZKPK for multiplication $\pi_{\mathsf{mul}}(d_1, d_2, d_3)$

This protocol is presented in [26] for integer commitments on groups with hidden order but it can easily be translated to our scenario.

The inputs of Prover and Verifier are the commitments $d_i = g^{x_i} h^{r_i}$ for $i = 1, 2, 3$. In addition, Prover is given the values x_i, r_i for $i = 1, 2, 3$. Obviously, both parties have access to the public parameters. In the following Σ-protocol, Prover aims to convince Verifier that $x_3 = x_1 x_2$. We can see that since $d_3 = d_1^{x_2} h^{r_3 - x_2 r_1}$ is a commitment on the value x_2 using d_1 as base instead of g, if Prover can convince Verifier that d_2 and d_3 commit on the same value and that he can open d_1, then the case is closed.

1. Prover chooses at random $x'_1, x'_2, r'_1, r'_2, r'_3 \in \mathbb{Z}_q$. Then Prover computes and sends $\mathfrak{a}_1 = g^{x'_1} h^{r'_1}$, $\mathfrak{a}_2 = g^{x'_2} h^{r'_2}$ and $\mathfrak{a}_3 = d_1^{x'_2} h^{r'_3}$ to Verifier.
2. Verifier randomly picks a challenge $\mathfrak{e}$ and sends it to Prover.
3. Prover computes and sends to Verifier: $\mathfrak{z}_{x_1} = x'_1 + \mathfrak{e} x_1$, $\mathfrak{z}_{x_2} = x'_2 + \mathfrak{e} x_2$, $\mathfrak{z}_{r_1} = r'_1 + \mathfrak{e} r_1$, $\mathfrak{z}_{r_2} = r'_2 + \mathfrak{e} r_2$ and $\mathfrak{z}_{r_3} = r'_3 + \mathfrak{e}(r_3 - x_2 r_1)$.
4. Finally Verifier checks that $g^{\mathfrak{z}_{x_1}} h^{\mathfrak{z}_{r_1}} = \mathfrak{a}_1 d_1^{\mathfrak{e}}$, $g^{\mathfrak{z}_{x_2}} h^{\mathfrak{z}_{r_2}} = \mathfrak{a}_2 d_2^{\mathfrak{e}}$ and that $d_1^{\mathfrak{z}_{x_2}} h^{\mathfrak{z}_{r_3}} = \mathfrak{a}_3 d_3^{\mathfrak{e}}$.

C.2 NIZKPK for interval membership $\pi_{\mathsf{ran}}(d, I)$ (range proof)

The main idea of the proof is to decompose a commitment into several commitments respectively to a binary decomposition. Then, the Prover computes a π_{or} on each commitment. The verification of the proof is the verification of each of the π_{or} and the recombination of the commitment decomposition. However the binary decomposition is a common possibility, there exists many refinements that optimize the cost of the proof when the intervals are not power of 2. The work of Canard et al. [27] provides an interesting comparison between the different methods as well as an original solution.

The inputs of Prover and Verifier is the commitment $d = g^x h^r$ and a binary decomposition base $b_0, \cdots, b_k$ where $b_i = 2^i$. In addition, Prover is given the values x, r where $x \in I = [0, 2^{k+1}[$. The value x can be written as $x = \sum b_i x_i$ where $x_i \in \{0, 1\}$ for $i = 0, \cdots, k$.

1. Prover chooses at random $r_i \in \mathbb{Z}_q$ for $i = 0, \cdots, k-1$ and computes $d_i = g^{x_i} h^{r_i}$ for $i = 0, \cdots, k$ where $r_k = (r - \sum_{i=0}^{k-1} r_i b_i) b_k^{-1}$. Prover sends each d_i to Verifier and then engages a $\pi_{\mathsf{or}}(d_i)$ with Verifier for $i = 0, \cdots, k$.
2. In addition to participating in each $\pi_{\mathsf{or}}(d_i)$, Verifier checks that $d = \prod_{i=0}^{k} d_i^{b_i}$. If each proof and the recomposition of the commitment are accepted, Verifier accepts the proof.

C.3 NIZKPK for comparison $\pi_<(d_1, d_2, d_3)$

This proof shows that $d_3 = d_1 < d_2$, in the sense that d_3 commits on 0 or 1 depending if the value committed in d_1 is strictly smaller than the value committed in d_2 or not. Of course this only make sense relatively to the binding property of the commitment scheme since d_1 and d_2 could theoretically open to any value in $\mathbb{Z}_q$. Furthermore, in a cyclic group, the notion of order is conceivable only on small intervals of $\mathbb{Z}_q$, typically centered in 0.

To achieve that proof, given d_1, d_2, one computes a $\pi_{\mathsf{ran}}(d_4,]-L, L[)$ where $d_4 = \mathsf{Com}(x_4) := \mathsf{Com}(x_1) - \mathsf{Com}(x_2)$. Indeed, if $x_1, x_2 \in [0, L[\subset \mathbb{Z}_q$ where $L < \lceil q/2 \rceil$ to avoid overlap, then $x_4 := x_1 - x_2 \in [-L, L[= [-L, 0[\cup [0, L[$. Given that $b_0, \cdots, b_k$ is the base for the binary decomposition of a value in $[0, L[$, we can add the extra base element $b_{k+1} = q - L$. Then x_4 admits a binary decomposition in this new base : $x_4 = \sum_{i=0}^{k+1} x_{4,i} b_i$ where $x_{4,i} \in \{0, 1\}$ for $i = 0, \cdots, k+1$. Moreover, $x_{4,k+1} = x_3$ thus $\pi_{\mathsf{ran}}(d_4, [-L, L[)$ provides the commitment d_3.

D Audit of the code.

Our implementation is meant to be used, but our setting allows the clients and the worker to develop their own code independently. To do so, the clients and the worker have to agree on the verification procedure that is used to verify the correctness of the function evaluation. For example, in the case of auctions, after that the worker has published the list of the sorted commitments with the π_{cor} proof, the verification proceeds in two steps: first, the clients verify that the sorted list of commitments is a permutation of the original list of commitments, then, the clients verify the π_{cor} proof which is an aggregation of $\pi_<$ proofs. However, other verification procedures are possible such as verifying the sorting algorithm itself. Aside from this, the worker needs to produce and send the public key pk of the CCE scheme to the clients. If someone wants to use our implementation or wants to rely on someone else's implementation, there are a few verifications to perform on the code. On the client's side, preventing privacy leakages means verifying that the code only encrypts his private information with the public key and sends only his ciphertext to the worker while publishing the commitment on the public board. Moreover, to obtain verifiability, the clients

must check that the code verifies the π_{cor} proof correctly. However, it is not important for the clients to audit the worker's code since the verifiability guarantee relies only on the π_{cor} proof and the privacy is, by hypothesis, entrusted to the worker.

E Implementation details

The CCE scheme of [22] is implemented through Elliptic Curves (EC). More precisely, we work with an optimized set of BN-curves [28, 29] which enables extension fields with nice properties. We denote $\mathbb{F}_p$ the prime field where p is a λ-bit prime. $E(\mathbb{F}_p)$ is the BN elliptic curve of equation $y^2 = x^3 + b$ where b is a well-chosen parameter [28]. In $E(\mathbb{F}_p)$, we pick P a generator of prime order q where $q = |E(\mathbb{F}_p)|$. The primes p and q are given by $p = p(u) = 36u^4 + 36u^3 + 24u^2 + 6u + 1$ and $q = q(u) = 36u^4 + 36u^3 + 18u^2 + 6u + 1$ for $u \in \mathbb{Z}$. The EC point P generates a group G_1 that will be used to compute the perfectly hiding commitments of the CCE scheme. We also consider $E'(\mathbb{F}_{p^2})$ the twisted elliptic curve over the extension field $\mathbb{F}_{p^2}$ in which we find the EC point Q, a generator of G_2, a prime order q group. The group G_2 will be used to compute the encryption part of the CCE scheme. As an extra feature, this setting offers an optimal ate pairing if one considers $G_3 \subset \mathbb{F}_{p^{12}}$, a q order group. We then define the non degenerate bilinear ate pairing $e : G_1 \times G_2 \to G_3$. This pairing is used in the decryption algorithm. Below are the detailed algorithms of the CCE scheme:

- $\mathsf{Gen}(1^\lambda)$: The public parameters pp contains the description of the groups G_1, G_2 and G_3, the pairing e and generators P, P_1 for G_1 and Q for G_2. Note that one must not know the discrete logarithm of P_1 in base P. Select randomly x from $\mathbb{Z}_q$ and compute $Q_1 = xQ$. The public key is $\mathsf{pk} := Q_1$ and the private key $\mathsf{sk} := x$. Return $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}_{\mathsf{pk}}(m)$: Pick at random $r, s \in \mathbb{Z}_q$ and compute $d := rP + mP_1$, $c_1 := sQ$, and $c_2 := rQ + sQ_1$. Return $c := (d, c_1, c_2)$.
- $\mathsf{Dec}_{\mathsf{sk}}(c)$: Parse c as (d, c_1, c_2) and compute $\tilde{m} := e(P, xc_1 - c_2)e(d, Q)$. Return the discrete logarithm of $\tilde{m}$ in base $e(P_1, Q)$.
- $\mathsf{DerivCom}_{\mathsf{pk}}(c)$: Parse c as (d, c_1, c_2) and return d.
- $\mathsf{Open}_{\mathsf{sk}}(c)$: Parse c as (d, c_1, c_2), compute and return $o := c_2 - xc_1$.
- $\mathsf{Verify}_{\mathsf{pk}}(d, o, m)$: Verify the equality $e(P, o) \stackrel{?}{=} e(d - mP_1, Q)$. If it holds return 1, otherwise return 0.

The ElGamal-like encryption part of the scheme in G_2 allows decryption for a small range of messages m (in our case, we set $m < 2^{16}$). However this is enough for our applications. We managed to use fast algorithms for scalar multiplication of EC points when precomputation is available [30] and when it is not [31]. We also rely on several optimizations to compute pairings when necessary [32–36].

Table 3. complexity and proof size for π_{cor} of applications - the linear system is performed with a coefficients matrix of size $m \times n$, each client possessing one coefficient. We consider auctions with $n+1$ bids of size $< 2^{k+1}$. The graph used for the shortest path contains m vertices and n weighted edges of weight $< 2^{k+1}$, each client possessing one weighted edge.

	Worker		**Client**	
	algorithm	preparing proof π_{cor}	$\lvert\mathcal{C}\rvert$	checking proof π_{cor}
Linear System Solving	$O(n^3)$	$[mn]\mathtt{Sm}$ $+[2mn]\mathtt{Sm_p}$ $+[2mn]\mathtt{A}$ proof size : $(n+2m)\mathtt{U}$	mn	$[2mn-1]\mathtt{Sm}$ $+[2mn+2m+6]\mathtt{Sm_p}$ $+[3mn+1]\mathtt{A}$
Auctions	$O(n \log n)$	$[(n+1)(3k-1)]\mathtt{Sm}$ $+[(n+1)(10k-1)$ $-6k+2]\mathtt{Sm_p}$ $+[(n+1)(7k-1)$ $-3k+1]\mathtt{A}$ proof size : $(n-1)(5k\mathtt{U}+4\mathtt{U}+k)$	n	$[2n(3k-1)]\mathtt{Sm}$ $+[n(6k+2)+6k+8]\mathtt{Sm_p}$ $+[n(8k-1)+3k+3]\mathtt{A}$
Shortest Path	$O(mn)$	$[n(3k-1)]\mathtt{Sm}$ $+[mn(6k+14)$ $+2m+3kn]\mathtt{Sm_p}$ $+[3k+1]\mathtt{A}$ proof size : $mn(5k\mathtt{U}+30\mathtt{U}+k)+8m\mathtt{U}$	n	$[mn(3k+8)$ $+(n-1)(3k-1)]\mathtt{Sm}$ $+[mn(3k+12)+8m$ $+(n-1)(3k)+6k+8]\mathtt{Sm_p}$ $+[mn(4k+18)+4m$ $+(n-1)(4k-1)+3k+3]\mathtt{A}$

F Complexity analysis of the test applications

In Table 3, we list the complexities to compute and verify the π_{cor} of the three applications we considered in the Section 4. Note that in the case of a linear system, we place ourselves in the simplest scenario where the worker provides as π_{cor}, a list of openings.

From the client's point of view, the complexities of the linear system and the auctions are linear in the number of clients. For the shortest path it equals the complexity of the Bellman-Ford algorithm.

G Miscellaneous

Exploring some other problems that adapt very well with our approach, our attention was caught in particular by the wide category of NP-problems. A first interesting one is the *graph coloring problem* where one wants to color each vertex of a graph given a fixed number of colours and where two adjacent vertices cannot have the same colour. In the multi-party case we assume that the structure of the graph is secretly shared among the clients. While this problem might be hard to solve, the solution when found could be easily verified.

Another graph NP-problem is finding an *Hamiltonian cycle* in a given graph. This kind of cycle is a path that visits each vertex exactly once. No known

algorithm computes the solution of this problem in polynomial time whereas verifying a given solution is linear in the number of nodes. Again, in multi-party, we assume that each client knows only a subset of the graph while the entire structure remains hidden.

Finally, other optimization problems might fit well with our technique. The *knapsack problem* is a good example with multi-party applications. For example, a set of clients share the use of a truck that has a maximum weight capacity. They all have a fixed number of items with respective secret weights. The goal is to maximize the number of items stored in the truck without exceeding its capacity. As this problem may not have a deterministic solution, the clients could require the worker to reach some minimal bound. Verifying that the bound is reached is then a matter of a simple sum.